NTNU
Kunnskap for en bedre verden

Department of Mathematical Sciences

TMA4285 - Time Series

Forecasting a cryptocurrency

Authors:

Martin Lie, Camille Loisel, Frederick Nilsen

October, 2021

Contents

List of Figures

List of Tables

1 Abstract

In this project, our goal was to use both time-series estimation, methods and machine learning to estimate and predict a chosen cryptocurrency stock. Due to the volatile characteristic of the stock, our estimates varied greatly as the data set on average does not satisfy the assumptions of stationary and normal distribution, which weakened our results and predictions. However, by further analysis, we found the series to satisfy a *heavy-tail distribution*, which shows why our predictions differed from the real values.

2 Introduction

The main motivation for this project is to be able to use time series analysis to forecast future values of the stock price of the cryptocurrency Litecoin. We start by exploring, plotting and generally commenting the data, followed by transformations of said data. Thereafter, we estimate the model parameters - with uncertainty and discuss model choice. Lastly, we use LSTM (long short-term memory), a widely used machine learning technique to forecast the same time series and compare the results.

2.1 Choice of dataset

We chose to explore the *Coinbase Litecoin* dataset due the comparatively few large changes appearing in the data from 2017 to 2021, compared with other similiar models for cryptocurrencies. As seen in Figure 1, we observe only two periods with large tops, while some form of seasonality s_t appears in the data (although small and seemingly insignificant), while the general trend m_t appears to be an increasing one. We therefore, for simplicity, make the following assumption regarding the data :

$$X_t = m_t + s_t + Y_t, \tag{1}$$

where X_t : the value of the stocks at a given time t. m_t is the **trend component** assumed to be a function of t and either decreasing or increasing, s_t is the **seasonality component**, again a function of t following a known period d, and Y_t is a random stationary noise component. We also assume that $\{Y_t\}$ is weakly stationary [2].

Remark. *It should be noted that our assumption of the model implies removing both trend- and seasonal components will help to get stationary residuals, but given the sharp increases in the data, using differencing or logarithmic transformation will further help to weaken the effect of these values.*

2.1.1 Brief history of the Litecoin

In short, Litecoin is a *peer-to-peer cryptocurrency* and an open-source software project [12], based on the same technology and similar to *Bitcoin*. Our assumptions are mainly based on the values and past history of the data. Seeing as we have a non-stationary series and periods with significant changes, this implies that normally distributed *residuals* for instance is not present (notice for instance two large sharp tops in Figure 1, which violates the finite variance σ when the error terms $\epsilon_i \sim N(0, \sigma^2)$). These periods are influenced by economical mechanisms and influences. In summary, we can list a few important periods and incidents that has shaped the historical value of the Litecoin [4]:

- Created by *Charlie Lee* in 2011, Litecoin was listed by *Coinbase* in May 2017 and started to grow exponentially. The latter part of 2017 and the beginning of 2018 was seen as the

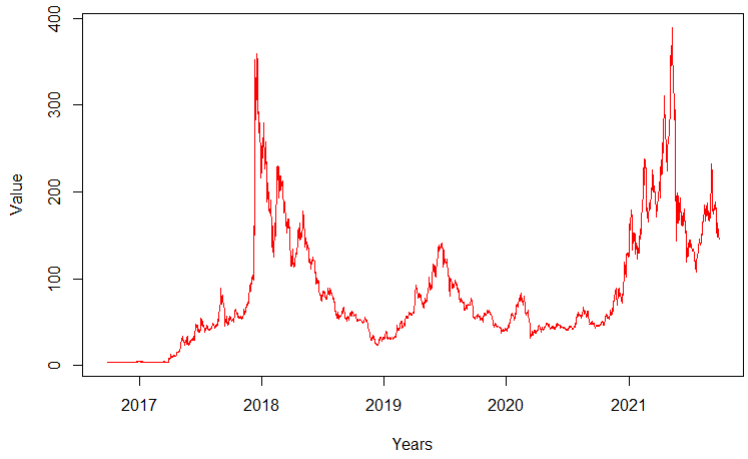

Figure 1: Litecoin values in between 2017 and 2021. Value is given in terms of US Dollars.

cryptocurrency bubble. As 2018 moved on, a crash appeared in the marked and the value fell.

- Part of this initial large growth came from the fact the currency in 2017 adapted the *Segregated Witness* protocol, which established the *Lightening Network*. This significantly increased the speed of transactions [1].
- The last significant period appeared in September of 2021, when the news-network *GlobeNewswire* released a fake statement saying that *Walmart* and Litecoin had established a partnership [8]. This caused a huge spike of 30% increase in the value.
- In the time period around June of 2019, Litecoin purchased $9,9\%$ of the conservative bank *WEG BANK* [7], which coincides with an upturn in value.
- The peak period around the turn of the year 2021 coincided with Bitcoins value dropping significantly due to Elon Musks and China blocking the currency being used in their related businesses [9], but also that Litecoin was finally accepted by *PayPal*, which sparked a renewed interest.

While analyzing the data, we will mainly be using the statistical software *R*, and *Python* for the *machine-learning* part. In *R*, we will mainly be using items from the *forecast* and *tseries* libraries. References to inbuilt-functions and methods will be used.

In order to analyze the data properly using the methods that shall be described, we need to make sure our data (or the transformed version) satisfies the conditions for stationary processes described in [13]. Deciding which method and model selection that will be used for further analysis will depend on the estimated *mean* $\overline{x}$ and *autocorrelation* $\hat{\gamma}$. Noticing how these values varies over time (or not) greatly impacts our progress and decisions made.

2.1.2 Chosen methods for the analysis

As indicated in Figure 1, certain days have extreme deviations compared to their neighbouring days, making it more difficult to model these appropriately according to the model assumption in (1). A simple remedy is to use differencing, going through the *classical decomposition method* proposed by [2].

Two approaches are viable here, where we can either try to estimate both m_t and s_t, or try to simply eliminate using the method mentioned. Trying to estimate both components on paper, adds

little value to our forecasting though is helps to evaluate the data on closer inspection. Using the *decompose*-function in R, we find that the seasonal component is non-existing, meaning $\nexists d \in \mathbb{N}$ s.t. $s_{t-d} = s_t$. From now on, we assume that $s_t = 0$, meaning that our model simplifies to a *non-seasonal model with trend* :

$$X_t = m_t + Y_t. \tag{2}$$

Using the first idea, we chose to start of by transforming the data using the ln()-operator. This helps to compress the values into a smaller window, thus making it more easier to further transform the data and using differencing. Another method is using polynomial fitting to our data in order to estimate m_t.

3 Theory

We present some vital theory used in the analysis of the time series in this section. We also refer to Appendix B for additional theory we were unable to fit in this section.

3.1 AR(I)MA model

In the statistical analysis of time series, *autoregressive-moving-average* (ARMA) model is a linear combination of two linear models, one for the *autoregression* (AR) and the second for the *moving average* (MA), and thus ARMA model is itself still linear. Given a time series of data $\{X_t\}$, the ARMA(p, q) model is a tool for understanding and predicting future values in this series. In short, AR(p) makes predictions using previous values of the dependent variable and MA(q) makes predictions using the series mean and previous errors [2].

3.1.1 AR(p) : Autoregressive model of order p

In an autoregressive model, we forecast the variable of interest using a linear combination of past values of the variable. The term autoregressive indicates that it is a regression of the variable against itself. The AR(p) model is generally written as :

$$X_t = Z_t + \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p}, \tag{3}$$

where $\{Z_t, t \in \mathbb{Z}\} \sim WN(0, \sigma^2)$. This is like a multiple regression but with lagged values of $\{X_t\}$ as predictors.

3.1.2 MA(q) : Moving average model of order q

Rather than using past values of the forecast variable in a regression, a moving average model uses past forecast errors in a regression-like model. The MA(q) model is expressed as :

$$X_t = Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q}, \tag{4}$$

where $\{Z_t, t \in \mathbb{Z}\} \sim WN(0, \sigma^2)$ and $\theta_1, \ldots, \theta_q$ are all constants.

3.1.3 ARMA(p, q)

A stationary process $\{X_t, t \in \mathbb{Z}\}$ is defined as an ARMA(p, q) process if :

$$X_t - \phi_1 X_{t-1} - \cdots - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q}, \tag{5}$$

for all $t \in \mathbb{Z}$, where $\{Z_t, t \in \mathbb{Z}\} \sim WN(0, \sigma^2)$.
For some $\phi_1, \ldots, \phi_p$ and $\theta_1, \ldots, \theta_q$, we have the relation [2] :

$$\phi(u) = 1 - \sum_{j=1}^{p} \phi_j u^j \tag{6}$$

$$\theta(u) = 1 + \sum_{j=1}^{q} \theta_j u^j \tag{7}$$

3.2 Recursive algorithms

For more general stationary processes, it would be helpful if the one-step predictor $P_n X_{n+1}$ based on n previous observations could be used to simplify the calculation of $P_{n+1} X_{n+2}$, the one-step predictor based on $n + 1$ previous observations. Prediction algorithms that utilize this idea are said to be recursive.
There is one important recursive prediction algorithm used in this project, namely the Innovations algorithm.

3.2.1 The Innovations algorithm

We will briefly present a definition of a Innovation Estimates of Moving Average parameters as described in [10]. If $\widehat{\gamma}(0) > 0$, we define the innovation estimates $\widehat{\theta_q}, \widehat{\sigma}^2$ appearing in (4) for $q = 1, 2, \ldots, n-1$, by the recursion relations,

$$\begin{aligned}
\widehat{\sigma}_0 &= \widehat{\gamma}(0) \\
\widehat{\theta}_{q,q-k} &= \widehat{\sigma}_k^{-1} \left[\widehat{\gamma}(q-k) - \sum_{j=0}^{k-1} \widehat{\theta}_{q,q-j} \widehat{\theta}_{k,k-j} \widehat{\sigma_j} \right], \quad k = 0, \ldots, q-1, \text{ and} \\
\widehat{\sigma}_q &= \widehat{\gamma}(0) - \sum_{j=0}^{q-1} \widehat{\theta}_{q,q-j}^2 \widehat{\sigma}_j
\end{aligned} \tag{8}$$

3.3 The augmented Dickey-Fuller test

An important step in our analysis process is to make our series stationary, but in order to identify whether or not our series fulfills that criteria we decide to use the *augmented Dickey-Fuller test* (ADF). The test is built upon the *Wald Statistic* which measures the likelihood-distance between an hypothesis H_0 and its alternative H_1. In such cases, one imposes the null-hypothesis criteria and lets the alternative one be unrestricted (usually by allowing it to be $\neq$ to value of H_0) [13]. The test itself is used to determine the presence of a unit root (of the $AR(p)$-process within the series), i.e. the null hypothesis suggests that the series is a *random walk* or a *casual process*. We

define

$$\gamma = \sum_{j=1}^{p} \phi_j - 1, \tag{9}$$

and furthermore the test statistic $t_\gamma = \frac{\hat{\gamma}}{se(\hat{\gamma})}$ (where se = standard error estimate of γ).

Should $\gamma = 0$, we then have that X_{t-1} provides no relevant information and the series does not revert back to the mean. *p-values* below 5% is used as a threshold to determine the series to be stationary [13].

4 Results

4.1 Results on the general model

While we generally utilized *R*'s built-in functions for finding the fit, we initially tried estimating the coefficients manually assuming our ARIMA-process was *casual*, meaning that

$$X_t = \sum_{j=0}^{\infty} \psi_j Z_{t-j}, \tag{10}$$

where the coefficients ψ_j is satisfied s.t.

$$\psi_j = \theta_j + \sum_{i=1}^{min(j,p)} \phi_i \psi_{j-i}. \tag{11}$$

We simply assume that $\theta_0 = 1$ and that $\theta_j = 0$ when $j > q$. In order to estimate the ψ_j, we utilize the innovations algorithm and find estimates $\hat{\theta}_{m,1}, \ldots, \hat{\theta}_{m,p+q}$. It follows that for large sample sizes n, and assuming that $E[Z_t^4] < \infty$, then the joint distribution of

$$n^{\frac{1}{2}}(\hat{\theta}_{m,1} - \theta_1, \ldots, \hat{\theta}m, k - \theta_k) \tag{12}$$

converges to a multivariate normal distribution with mean $\mathbf{0}$ and covariance A [2] with elements

$$a_{ij} = \sum_{r=1}^{min(i,j)} \theta_{i-r}\theta_{j-r}. \tag{13}$$

Then in order to find the coefficients for the ARIMA-process, we replace the ψ_j in equation (11) with the $\hat{\theta}_{m,j}$ s.t. we need to solve the following equations for $j = 1, \ldots, p + q$.

$$\hat{\theta}_{m,j} = \theta_j + \sum_{i=1}^{min(j,p)} \phi_i \hat{\theta}_{j-i}. \tag{14}$$

Our estimates, as shown in Table 1, proved out to be close, but still to far off in order to make a good predictions as possible. We use the normal difference analysis as an example.

4.1.1 Estimating m_t and removing it from the set

In order to find the best trend to our model, we decide to assume that the trend component is an approximate polynomial, i.e.

$$m_t(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n. \tag{15}$$

We therefore repeatedly used the *AIC-criterion* (B.2) in order to find the polynomial with the fewest degrees and lowest AIC-value. Running through degrees up to 30 we found the best trend fit for $n = 21$. Removing m_t from our set, gave us a stationary series according to the ADF-test. Using our *ARIMA*-estimation through the *auto.arima*-function in *R*, we found that $AR(1)$-process fitted our data best, see Table 2.

4.1.2 Using logarithmic operation

We then used the *natural logarithm* on the stock values to compress them into a more stable and stationary series, confirmed by the ADF-test. This method ended up giving us an $ARIMA(1, 1, 1)$-process as the best fit, see Table 3.

4.1.3 Normal analysis using differencing

Knowing the presence of a trend term m_t, we decided to simply use differencing once to eliminate m_t, and our analysis gave us that an $ARIMA(1, 1, 4)$-process was the best choice, summarized in Table 4.

4.2 LSTM model

For the LSTM model, we use about 72 % of the valid data for training, 18 % for validation and the latter 10 % for testing the model. The sequential model has 4 LSTM layers with units 50, 60, 80 and 120 and all with the ReLU[1] activation function. We also have a dropout layer of 0.2 and a last dense layer.

Training this model with the Adam-algorithm at batch sizes of 128 over 20 epochs, we end up with a loss of about 0.0013, as shown in the loss development in Figure 3.

The result over all test data is highlighted in Figure 2. We may also crop the results to the latest 21 entries only to gain comparative results to the time series forecasting. This results in Figure 4. The mean error over these values is about 28.57.

5 Discussion

We sum up the results and predictions on the latest 21 entries being added to the data. Our final results can be summed in Table 5.

5.1 Discussion on time-series methods

The general assumption being made in our analysis was that our sample data came from an (approximate) Gaussian distribution. But comparing our residuals for our models to the real data, we found that this was not the case. In all three cases, the distributions was Cauchy as highlighted in Figure 5. As previously mentioned, this is a heavy-tail distribution, which explains why our data

[1]Rectified linear unit: A piecewise linear function that is the identity for positive values and zero otherwise [3].

has extreme values and changes. This is connected to the aforementioned history of the Litecoin from Section 2.1.1. The Litecoin is a volatile stock and easily influenced by the outside world. Therefore, assuming a finite variance σ is not realistic.

Comparing the three models to each other, we see that the logarithmic prediction on average was the best one, closely followed by the differencing-model. While removing the trend seemed to create a stationary series in the first model, our estimate of m_t was unable to capture the extreme values of the series. Moreover, our plots for both the estimated ACF and $PACF$ in Figure 6 and 7 did not follow the characteristics of an AR-model. In general, both plots had not apparent trend or pattern meaning that the data most likely comes from an ARMA-process. Finally, by estimating m_t up to such a high degree, it most likely that we also overfitted the data, giving us a poor prediction as a result. Also, this model has the highest values for the model estimation, confirming that this model is the least efficient model out of the three. While the third model does lead to a low estimation error in regards to the prediction, the main issue is that the model estimation is in the same order as the trend-model. Therefore, the logarithmic transformation seems like the best fit. By compressing the data, our model was made into an approximate stationary process, as the data now does not exhibit the huge fluctuations in the data [2]. In summary, a better model for the dataset would be to use $GARCH$-models which takes into account how volatile the series becomes and the heavy-tailness of the data [2].

5.2 Discussion of LSTM results

Using known machine learning packages in Python like Keras and TensorFlow, we are able to construct a LSTM network as described in Section A. As shown in the results, Figure 2 shows the real versus predicted stock value of Litecoin. In general, the trained network is able to predict the general curvature of the model well.

Although the model also recognizes spikes, it is not as extreme as the actual values. Potential reasons for this could either be overfitting, or that the trained dataset had too little variation in its set. Previous attempts with fewer layers and epochs revealed however a generally worse fit, which rules out overfitting as an issue. Studying the time series from Figure 1 however, we see that the first months of Litecoin's lifespan stayed at about the same, low value. Since this is not representative of the time series after the cryptocurrency gained its popularity, training the model from about the summer of 2017 may remedy the LSTM-model's difficulty with dealing with spikes. This may also be a factor contributing to the general observation that the predicted values seem to be lower than the actual values.

Another note worth mentioning, is that the data is normalized with min-max-normalization which doesn't cope with the variance like a Cauchy scaling could. However, attempting non-standard scalers and attempting to apply external libraries like TensorFlow will generally introduce more problems that it alleviates, prompting us to not explore such a solution.

6 Conclusion

Both the ARMA- and LSTM-modeling provided reasonably accurate forecasts for some of the future values. Its accuraccy is debatable in both cases, but the ARMA-modeled projections managed to find the future stockvalues within its estimated variance. However, all models struggled with the data being Cauchy(heavy tail)-distributed. With infinite variance due to the volatile nature of the Litecoin-stock our predictions struggled to capture the true essence of the data. For further analysis, more advanced models such as $GARCH$-models in order to accommodate for the volatility.

Appendix

A Short on RNNs and LSTM

Although the main focus on this project is not machine learning, it is of interest to gain a baseline understanding of the mechanics of Recurrent Neural Networks (RNNs) and especially LSTM-networks. Seeing as this was not required in the project, it is placed in the appendix.

One of the main components of a RNN is its sequential memory. It has many applications, like in speech, text, audio and financial data as we are exploring in this project. In general, neural networks works by propagating inputs through a set of hidden layers and at last return it to the output layer. RNNs implement sequential memory, essentially by looping the previous iterates in the hidden layers, allowing previous information to influence the output of future layers [11].

LSTM-networks, i.e. long short-term memory networks, are a sub-genre of recurrent neural networks. By using forget-gates, they determine which data is important to keep and forgets the unimportant data. The net input in LSTM cells consists of the actual input gate, the *forget gate* and the *output gate* from previous iterations. The three gates collect activations from inside and outside the block. The input and output gates scale the input and output of the cell while get forget gate scales the internal state - e.g. by resetting its weight 0 [5].

B Additional theory

We present some additional material that in short discusses the theory related to *heavy-tail distribution* and model selection.

B.1 Heavy-tail distribution

An important assumption being made in the previous section was the fact that our observations from the ARMA-model was assumed to be *Gaussian* distributed [2]. This however, assumes that $\sigma < \infty$ and within financial statistics, this is not always the case. Therefore, we introduce the concept of *heavy-tail distributions*. Mostly used for rare events, these distributions are used to model events where extreme values occurs with a high probability, for example within finance [6]. We generally have that for some $\alpha > 0$,

$$\lim_{n \longrightarrow \infty} P\{\frac{\max(X_1, \dots, X_n) - b_n}{a_n} \leq x_n\} = e^{-(1+\frac{x}{\alpha})^{-\alpha}}. \tag{16}$$

This means that the maximum value of the sample has an asymptotically extreme value distribution, and these types of distributions have no endpoints, meaning that for instance the variance is infinite for distributions. One of these, is the *Cauchy-distribution*.

B.2 Akaikes information criteria (AIC), AICC and BIC

When working finding the best model and size of both p and q, we wish to find the smallest number of parameters that fits our data, but does not overfit it. To protect against this happening, we use *Akaikes criterion* (AIC), the *bias-corrected AIC* (AICC) and the *Bayesian information criterion*. The first criterion is built on the idea of minimizing the Kullback-Leibner distance between the *true* model and the *candidate models*. Assuming our observations comes from an ARMA(p,q)-process with unknown parameters $\theta = (\beta, \sigma^2)$, the true model can only be indetified if all candidate models were testet which is an infinite number. Instead, we estimate the discrepanicy and find the model with the lowest estimated distance between the models [2]. This gives us the following definitions. We use that k is the total number of parameters, n the toal sample size and $\hat{L}$ is the maximum likelihood of the *null-hypothesis* model L:

$$\text{AIC} = 2k - 2ln(\hat{L}) \tag{17}$$

$$\text{AICC} = \text{AIC} + \frac{2k^2 + 2k}{n - k - 1} \tag{18}$$

The BIC-criteria assumes data from an zero-mean casual ARMA(p,q)-process and is given as

$$\text{BIC} = kln(n) - 2ln(\hat{L}). \tag{19}$$

The advangtage here is that, for $n \longrightarrow \infty$, then $\hat{p} \longrightarrow p$ and $\hat{q} \longrightarrow q$ and BIC decreases to zero. This makes BIC a consistent order-selection criteria [2].

C Figures and tables

ARIMA(1,1,4)-process	ϕ_1	θ_1	θ_2	θ_3	θ_4
Estimated values by *auto.arima*	-0.5353	0.5137	-0.0372	-0.0061	0.1252
Estimated values manually	-0.5014	0.4509	-0.064	-0.0111	0.1237

Table 1: Estimated coefficients of the ARIMA(1,1,4) process using manual calculations versus the R *auto.arima* function

AR(1)-process	ϕ_1
Estimated value	0.9543
Estimated standard error	0.0074
Model estimation	AIC: 12882.52 AICC: 12882.52 BIC: 12893.54

Table 2: Model estimation results for the AR(1) process

ARMA(1,1,1)-process	ϕ_1	θ_1
Estimated values	-0.8116	0.7763
Estimated standard error	0.0827	0.0884
Model estimation	AIC: -4973.4 AICC: -4973.39	BIC: -4956.87

Table 3: Parameter results after logarithmic transformation

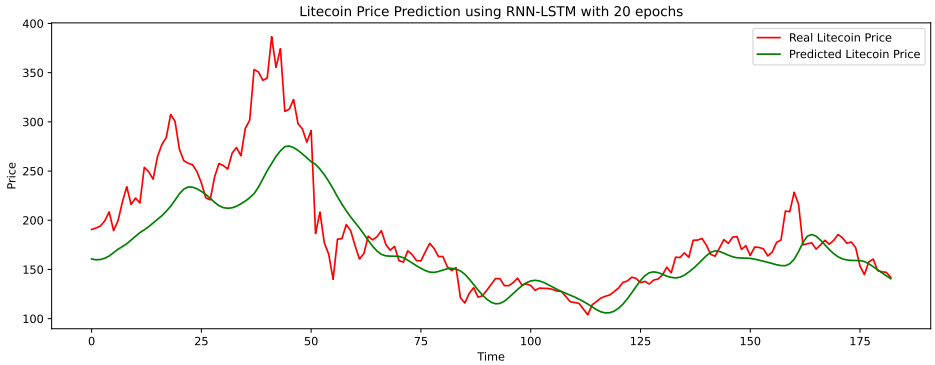

Figure 2: LSTM-predicted price development of Litecoin over the test dataset, i.e. March 29th 2021 until September 27th 2021. The ragged, red line shows the actual price for these days whereas the green line shows the predicted value for the same dates. Each time step is one day and the x-axis consists of the indices of the time span [2021-03-29, 2021-09-27].

ARIMA(1,1,4)-process	ϕ_1	θ_1	θ_2	θ_3	θ_4
Estimated values	-0.5353	0.5137	-0.0372	-0.0061	0.1252
Estimated standard error	0.0821	0.0832	0.0266	0.0246	0.0250
Model estimation	AIC: 12830.85	AICC: 12830.9	BIC: 12863.91		

Table 4: Parameter and model estimates after differencing.

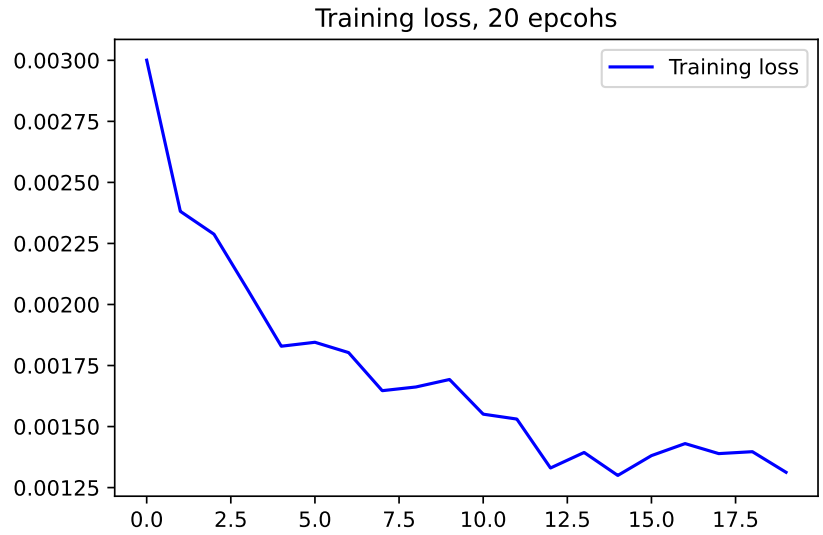

Figure 3: Training loss over each epoch of the LSTM network.

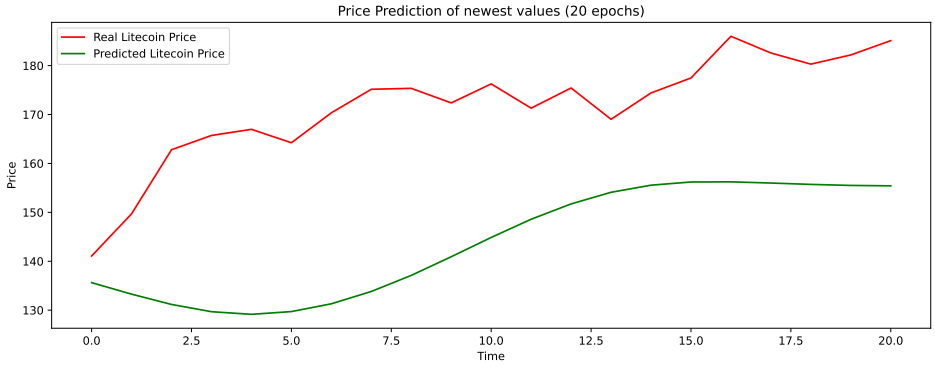

Figure 4: Actual versus LSTM-projected price for the latest 21 days available.

Prediction method	Mean error	95%-interval deviance
Trend removing	94.5282	(41.022,70.104)
Logarithmic transformation	28.66804	(15.662,43.335)
Differencing	29.63849	(17.688,43.073)
LSTM	28.56893	

Table 5: Summary of model fit with the different types of models.

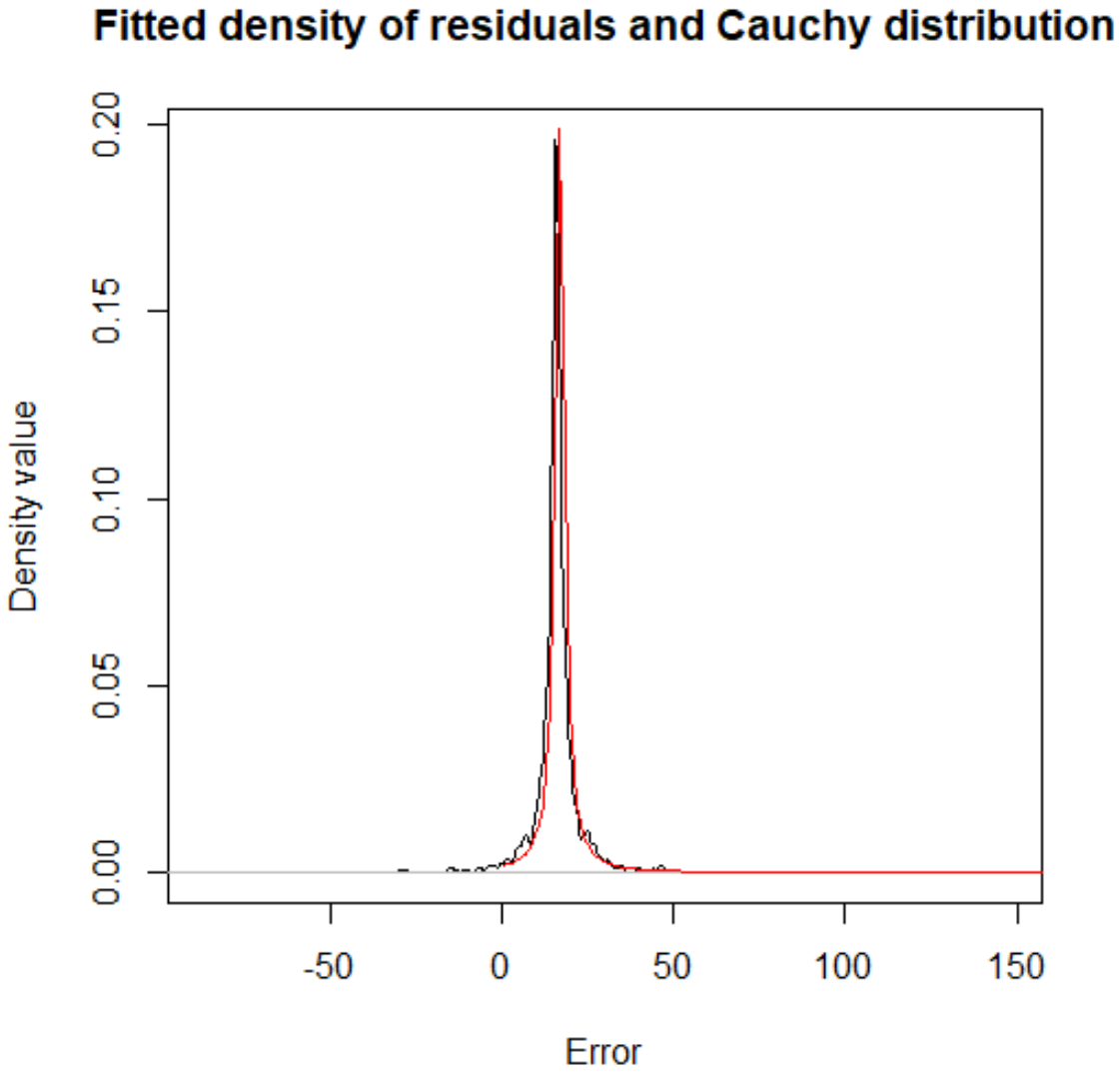

Figure 5: Fitted density of the model residuals compared to the Cauchy distribution. We use scale 1.6 and location 16 for our model

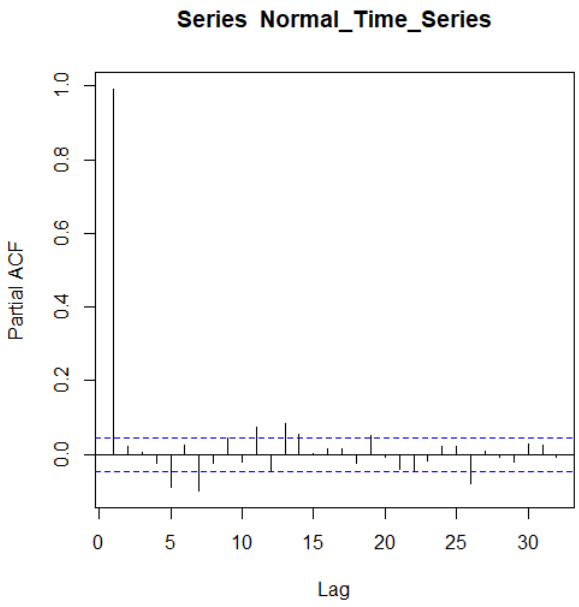

Figure 6: PACF Arma

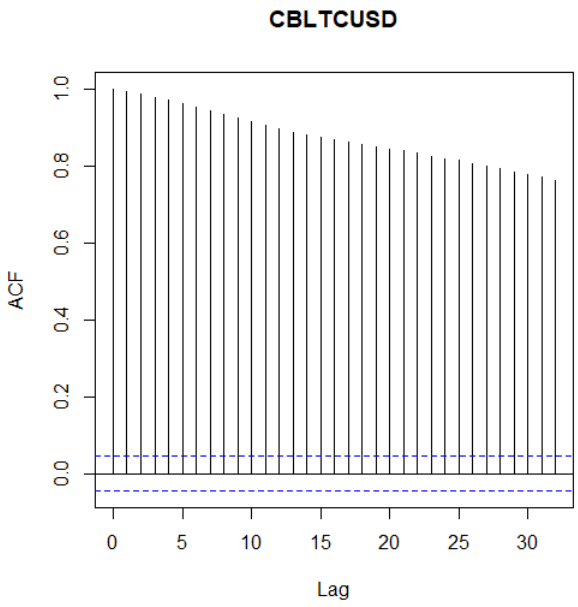

Figure 7: ACF Arma

D R code

Logarithmic-transformation with original dataset used in the analysis

```
label={lst:logtransform}]
library(readxl)
Data <- read_excel("CBLTCUSD.xls",
                        col_types = c("date", "numeric", "numeric"))
View(Data)

#Plotting the data
plot(Data$observation_date,Data$CBLTCUSD, type = "l", col = "red", xlab = "Years",
    ylab = "Value")
plot(Data$observation_date,Data$`Daily change`, type = "l", col = "red", xlab = "
    Years", ylab = "Daily changes")

#Finding outliers/abnormalities
AbsChange <- abs(Data$`Daily change`)
average_change <- sum(AbsChange, na.rm = TRUE)/as.numeric(length(AbsChange))
plot(Data$observation_date,abs(Data$`Daily change`), type = "l", col = "green",
    xlab = "Years", ylab = "Daily changes")
abline(h = average_change, col = "red")
var_abschange <- sqrt(var(AbsChange, na.rm = TRUE))
limit <- average_change + var_abschange
plot(Data$observation_date,abs(Data$`Daily change`), type = "l", col = "brown",
    xlab = "Years", ylab = "Daily changes")
abline(h = limit)
limit_ratio <- sum(AbsChange > limit, na.rm = TRUE)/as.numeric(length(AbsChange))
outliers <- which(AbsChange > limit)
AbsChange_new <- AbsChange[-outliers]
plot(c(1:length(AbsChange_new)),AbsChange_new, type = "l", col = "brown", xlab = "
    Index", ylab = "Daily changes")
New_Dates <- Data[-outliers,1]
Index <- c(1:as.numeric(dim(New_Dates)[1]))
New_Prices <- Data[-outliers,2]
plot(length(Index), as.matrix(New_Prices), xlab = "Index", ylab = "Value", type = "
    l", col = "orange")

#Log-transforming the data
LogAbsChange <- abs(log(abs(Data$`Daily change`)))
LogAbsChange <- LogAbsChange[!is.infinite(LogAbsChange)]
```

```r
log_average_change <- sum(LogAbsChange, na.rm = TRUE)/as.numeric(length(
    LogAbsChange))
plot(Data$observation_date,log(abs(Data$`Daily change`)), type = "l", col = "
    deeppink", xlab = "Years", ylab = "Log Daily changes")
abline(h = log_average_change, col = "red")
log_var_abschange <- sqrt(var(LogAbsChange, na.rm = TRUE))
log_limit <- log_average_change + log_var_abschange
plot(Data$observation_date,log(abs(Data$`Daily change`)), type = "l", col = "blue",
     xlab = "Years", ylab = "Log Daily changes")
abline(h = log_limit)
log_limit_ratio <- sum(LogAbsChange > log_limit, na.rm = TRUE)/as.numeric(length(
    LogAbsChange))
outliers <- which(LogAbsChange > log_limit)
LogAbsChange_new <- LogAbsChange[-outliers]
plot(c(1:length(LogAbsChange_new)),LogAbsChange_new, type = "l", col = "orange",
    xlab = "Index", ylab = "Log Daily changes")
LogNew_Dates <- Data[-outliers,1]
LogIndex <- c(1:as.numeric(dim(LogNew_Dates)[1]))
LogNew_Prices <- log(Data[-outliers,2])
plot(LogIndex,as.matrix(LogNew_Prices), xlab = "Index", ylab = "Log Price", type =
    "l", col = "red")

#Plotting the data log
plot(Data$observation_date,log(Data$CBLTCUSD), type = "l", col = "red", xlab = "
    Years", ylab = "Value")
log_model <- ts(log(Data$CBLTCUSD))
#Notice we have a NA-value in Time_Series
missing_value_index_log <- which(is.na(log_model))
# Set value equal to the day before
log_model[missing_value_index_log] <- log_model[missing_value_index_log-1]
acf(log_model)
pacf(log_model)
model_log <- auto.arima(log_model)
summary(model_log)
model_log_forecast <- forecast(model_log, h=22)
plot(model_log_forecast)

#Comparing with updated values
Updated_Data <- read_excel("CBLTCUSD_updated.xls",
                           col_types = c("date", "numeric"))
log_value_updated <- log(Updated_Data$CBLTCUSD)
length_update <- dim(Updated_Data)[1]
length_start <- length_update - 21
lines(c(1:length_update), log_value_updated, type = "l", col = "brown")
error2 <- mean(abs(Updated_Data$CBLTCUSD[length_start:length_update]-model_log_
    forecast$mean))
```

Trend and normal-analysis

```r
library(readxl)
library(tseries)
library(forecast)
Data <- read_excel("CBLTCUSD.xls", col_types = c("date", "numeric", "numeric"))
Updated_Data <- read_excel("CBLTCUSD_updated.xls", col_types = c("date", "numeric")
    )
#Plotting the data
plot(Data$observation_date,Data$CBLTCUSD, type = "l", col = "red", xlab = "Years",
    ylab = "Value")
plot(Data$observation_date,Data$`Daily change`, type = "l", col = "red", xlab = "
    Years", ylab = "Daily changes")
Normal_Time_Series <- ts(Data[,2])
start <- length(Normal_Time_Series) +1
end <- length(Normal_Time_Series)+22

#Trend component estimation by lowest AIC-value
poly_interval <- c(2:30)
aic_matrix <- matrix(nrow = length(poly_interval), ncol = 1)
for (i in 1:length(poly_interval)){
  model <- glm(CBLTCUSD ~ poly(observation_date,i), data = Data)
  aic_matrix[i,1] <- model$aic
}
best_model <- which(aic_matrix == min(aic_matrix, na.rm = TRUE))[1]
```

```
best_trend <- glm(CBLTCUSD ~ poly(observation_date,best_model+1), data = Data)
summary(best_trend)
plot(predict.glm(best_trend), xlab = "Time", ylab = "Predikert verdi vs. reell
    verdi", type = "l", col = "red")
lines(c(1:1827),Data$CBLTCUSD, type = "l", col = "blue")
#Even removing the trend component does not make our series stationary
plot(c(1:1827),(Data$CBLTCUSD-predict.glm(best_trend)), type = "l", col = "green")
without_trend <- ts(Data$CBLTCUSD-predict.glm(best_trend))
#Notice we have a NA-value in Time_Series
missing_value_index_trend <- which(is.na(without_trend))
# Set value equal to the day before
without_trend[missing_value_index_trend] <- without_trend[missing_value_index_trend
    -1]
adf.test(without_trend)
steps_trend <- ndiffs(without_trend)
model_without_trend <- auto.arima(without_trend)
summary(model_without_trend)
trend_forecast <- forecast(model_without_trend, h = 22)
plot(trend_forecast, type = "l", col = "blue")
error3 <- mean(abs(as.numeric(Updated_Data$CBLTCUSD[start:end])-trend_forecast$mean
    ), na.rm = TRUE)
error3_lower <- mean(abs(as.numeric(Updated_Data$CBLTCUSD[start:end])-trend_
    forecast$lower[2]), na.rm = TRUE)
error3_upper <- mean(abs(as.numeric(Updated_Data$CBLTCUSD[start:end])-trend_
    forecast$upper[2]), na.rm = TRUE)

#Notice we have a NA-value in Time_Series
missing_value_index <- which(is.na(Normal_Time_Series))
# Set value equal to the day before
Normal_Time_Series[missing_value_index] <- Normal_Time_Series[missing_value_index
    -1]
#ACF- and PACF-plot of the time series
acf(Normal_Time_Series)
pacf(Normal_Time_Series)
#Noticing that the value decreases slowly. This implies that an ARMA-model is the
    correct method
adf.test(Normal_Time_Series)
#The p-value from this test is larger than 0.05, so the series is non-stationary.
#The test implies the lag for the series is 12
#Checking how many differences is needed to make the series stationary
steps <- ndiffs(Normal_Time_Series)
Normal_Series_Diffed <- diff(Normal_Time_Series)
#Differencing the ts to make it stationary
#Normal_Series_Diffed <- diff(Normal_Time_Series,steps)
model_normal <- auto.arima(Normal_Time_Series)
summary(model_normal)
plot(model_normal) #Shows us the roots of the AR- and MA-processes
#Forecasting for the 100 next values
forecast_normal <- forecast(model_normal, h = 22)
plot(forecast_normal, type = "l", col = "green")
lines(c(1:end),Updated_Data$CBLTCUSD, type = "l", col = "red")
error1 <- mean(abs(forecast_normal$mean-Updated_Data$CBLTCUSD[start:end]))
error1_upper <- mean(abs(forecast_normal$upper[2]-Updated_Data$CBLTCUSD[start:end])
    )
error1_lower <- mean(abs(forecast_normal$lower[2]-Updated_Data$CBLTCUSD[start:end])
    )

density_residuals <- density(model_normal$residuals+16) #Shifts the values to the
    righ in order to fit the distribution
plot(density_residuals, xlab = "Error", ylab = "Density value", main ="Fitted
    density of residuals and Cauchy distribution")
lines((dcauchy((0:1827), location = 16, scale = 1.6)), col = "red") #Shows us that
    the errors are Cauchy-distributed

back_step <- matrix(0,1,length(Normal_Time_Series)+22)
for(j in 1:length(Normal_Series_Diffed)){
 back_step[j] <- Normal_Series_Diffed[j]
}

for(i in start:end){
back_step[i]<- forecast_normal$mean[i-length(Normal_Time_Series)]
}
back_step_ts <- ts(back_step)
```

```r
final_back_step <- diffinv(back_step_ts, differences = steps)
plot(c(1:end),final_back_step[2,], xlab = "Index", ylab = "Value", type = "l", col = "blue")

#Testing the model against updated values/comparing the forecasting to the real data


#Simulating from the model assuming the residuals follows a Cauchy distribution with location 0 and scale around 5
plot(arima.sim(n = 1827, list(ar = c(0.2403497, 0.5563574), ma = c(0.712329, 0.0904316, 0.08005933, 0.2755035)), rand.gen = rcauchy, n.start = 100, start.innov = rcauchy(100, scale = 5, location = 0)))
n <- 1000
h <- 100
simulations <- matrix(ncol = n, nrow = h)
simulations_mean <- matrix(ncol = 1, nrow = h)
for(i in 1:n){
  simulations[,i] <- arima.sim(n = h, list(ar = c(0.2403497, 0.5563574), ma = c(0.712329, 0.0904316, 0.08005933, 0.2755035)), rand.gen = rcauchy, n.start = NA, start.innov = rcauchy(100, scale = 5, location = 0))
}
for (j in 1:h){
  simulations_mean[j] <- Normal_Time_Series[start-1] + mean(simulations[j,])
  if(simulations_mean[j] < 0){
    simulations_mean[j] <- 0
  }
}
plot(simulations_mean, xlab = "The next h = 100 steps", ylab = "Predicted value", type = "l", col = "blue")

#Stolen content from Jorge
## MAtoARMA function made for finding thetas in equation (5.1.25) of the book

MAtoARMA <- function(psi,p,q)
{
  Psi.qp <- matrix(0,p,p)
  for( i in 1:p)
    for(j in 1:p)
    {
      if(q+i-j > 0)
      {
        Psi.qp[i,j] <- psi[1+q+i-j]
      } else if(q+i-j == 0){
        Psi.qp[i,j] <- 1
      }
    }
  phi <- as.numeric(solve( Psi.qp) %*% psi[1+(q+1):(q+p)])
  psi.0 <- c(rep(0,p-1),psi)
  theta <- numeric()
  for( j in 1:q)
  {
    theta[j] <- psi.0[p+j] - sum( phi[1:p] * psi.0[(p+j-1):j] )
  }
  output <- list( phi = phi,
                  theta = theta)
  return(output)
}

# 1 Get sample autocovariances from the series
n <- length(Normal_Time_Series)
gamma.hat <- my.acf(Normal_Series_Diffed,max.lag=n)$gamma.hat
K.hat <- matrix(NA,n+1,n+1)
for(j in 1:(n+1))
  for(i in 1:(n+1))
  {
    K.hat[i,j] <- c(gamma.hat)[1+abs(i-j)]
  }
# 2. Run the innovations algorithm on centered data
## We want the thetas to use them as input for MA to ARMA function
innov.hstep.out <- innov.hstep(Normal_Series_Diffed-mean(Normal_Series_Diffed, na.rm = TRUE),h=1,K.hat)
Psi <- innov.hstep.out$Theta
```

```r
# 3. Extract the values of interest to solving equation 5.1.25
## See explanation in page 136 after equation (5.1.25)
p <- 1
q <- 4
Psi.hat.mat <- matrix(NA,floor(n/2),p+q) ##Has to see with remark 2 in page 133
for(i in 1:floor(n/2))
{
  ind <- i:(i+p+q-1)
  Psi.hat.mat[i,] <- Psi[p+q+i,ind]
}
# plot columns of Psi.hat.mat against the value of m

plot(as.ts(Psi.hat.mat),xlab="m")

# choose m given the values that changes the least amount
best_m_matrix <- matrix(nrow = floor(n/2), ncol = 5)

for (i in 1:(floor(n/2))-1){
  for (j in 1:5){
    best_m_matrix[i,j] <- abs(Psi.hat.mat[i+1,j]-Psi.hat.mat[i,j])
  }
}
mean_matrix <- matrix(nrow = floor(n/2)-1, ncol = 1)
for (i in 1:(floor(n/2))-1){
  mean_matrix[i,1] <- mean(best_m_matrix[i,])
}
m <- which(mean_matrix <= min(mean_matrix))
psi.hat <- c(1,Psi.hat.mat[m,(p+q):1])


# 5. Solve the equations to find the estimates for the coefficients
MAtoARMA(psi.hat,p,q)
model_normal$coef
```

Innovation Algorithm

```r
innov.hstep <- function (X, h, K)
{
  X.cent <- X - mean(X) ## Zero-mean time series
  n <- length(X)
  v <- numeric(n + h) ##MSE of one-step prediction
  X.pred <- numeric(n + h)
  Theta <- matrix(0, n + h, n + h) ##Coefficients of linear combinations of
    innovations
  # The algorithm as described in the book
  v[1] <- K[1, 1]
  X.pred[1] <- 0
  Theta[1 + 1, 1] <- K[2, 1]/v[1]
  v[2] <- K[2, 2] - Theta[1 + 1, 1]^2 * v[1]
  X.pred[2] <- Theta[1 + 1, 1] * X[1]
  for (k in 2:n) {
    Theta[1 + k, k] <- K[k + 1, 1]/v[1]
    for (j in 1:(k - 1)) {
      Theta[1 + k, k - j] <- (K[k + 1, j + 1] - sum(Theta[1 +
                                                          j, j:1] * Theta[1 + k,
    k:(k - j + 1)] * v[1:j]))/v[j +
                                     1]
    }
    v[k + 1] <- K[k + 1, k + 1] - sum(Theta[1 + k, k:1]^2 *
                                        v[1:k])
    ##One step prediction ##
    X.pred[k + 1] <- sum(Theta[1 + k, 1:k] * (X.cent[k:1] -
                                                X.pred[k:1]))
  }
  if (h > 1) {
    for (k in (n + 1):(n + h - 1)) {
      Theta[1 + k, k] <- K[k + 1, 1]/v[1]
      for (j in 1:(k - 1)) {
        Theta[1 + k, k - j] <- (K[k + 1, j + 1] - sum(Theta[1 +
                                                              j, j:1] * Theta[1 + k
    , k:(k - j + 1)] * v[1:j]))/v[j +
```

```
                                        1]
      }
      v[k + 1] <- K[k + 1, k + 1] - sum(Theta[1 + k, (k -
                                                n + 1):k]^2 * v[n:1])
      ## Note: For h>1 the prediction uses the n innovations
      X.pred[k + 1] <- sum(Theta[1 + k, (k - n + 1):k] *
                             (X.cent[n:1] - X.pred[n:1]))
    }
  }
  for (k in 1:(n + h - 1)) {
    Theta[1 + k, 1:k] <- Theta[1 + k, k:1]
  }
  X.pred <- X.pred + mean(X)
  output <- list(X.pred = X.pred, v = v, Theta = Theta[1:n,
                                                       1:n])
  return(output)
}
```

Using logarithmic transformation on the data

```
#Plotting the data log
plot(Data$observation_date,log(Data$CBLTCUSD), type = "l", col = "red", xlab = "
    Years", ylab = "Value")
log_model <- ts(log(Data$CBLTCUSD))
#Notice we have a NA-value in Time_Series
missing_value_index_log <- which(is.na(log_model))
# Set value equal to the day before
log_model[missing_value_index_log] <- log_model[missing_value_index_log-1]
acf(log_model)
pacf(log_model)
model_log <- auto.arima(log_model)
summary(model_log)
model_log_forecast <- forecast(model_log, h=22)
plot(model_log_forecast)

#Comparing with updated values
log_value_updated <- log(Updated_Data$CBLTCUSD)
length_update <- dim(Updated_Data)[1]
length_start <- length_update - 21
lines(c(1:length_update), log_value_updated, type = "l", col = "brown")
error2 <- mean(abs(Updated_Data$CBLTCUSD[length_start:length_update]-exp(model_log_
    forecast$mean)))
error2_lower <- mean(abs(Updated_Data$CBLTCUSD[length_start:length_update]-exp(
    model_log_forecast$lower[2])))
error2_upper <- mean(abs(Updated_Data$CBLTCUSD[length_start:length_update]-exp(
    model_log_forecast$upper[2])))
```

Define autocorrelation function

```
# define autocorrelation function (Definition 1.4.4)
my.acf <- function(x,max.lag=12)
{
  n <- length(x)
  x.bar <- mean(x)
  gamma.hat <- numeric(max.lag+1)
  for(h in 0:min(max.lag,n-1))
  {
    gamma.hat[h+1] <- 0
    for(t in 1:(n-h))
    {
      gamma.hat[h+1] <- gamma.hat[h+1] + (x[t] - x.bar)*(x[t+h] - x.bar)
    }
  }
  gamma.hat <- gamma.hat / n
  rho.hat <- gamma.hat / gamma.hat[1]
  output <- list( gamma.hat = gamma.hat,
                  rho.hat = rho.hat,
                  lags = 0:max.lag)
  return(output)
}
```

Define a function to give the coefficients of a truncated MA(inf) representation of the ARMA(p, q) series.

```
# first define a function to give the coefficients of
# a truncated MA(inf) representation of the ARMA(p,q) series.
ARMAtoMAinf <- function(phi=NULL,theta=NULL,trun=500)
{
  if(length(phi)==0)
  {
    q <- length(theta)
    psi <- numeric(trun)
    psi[1:(q+1)] <- c(1,theta)
  } else if(length(phi)>0)
  {
    # check to see if the time series is causal:
    minroot <- min(Mod(polyroot(c(1,-phi))))
    if( minroot < 1)
      stop("The ARMA process specified is not causal.")
    p <- length(phi)
    q <- length(theta)
    # set theta_j = 0 for j > q
    theta.0 <- c(theta,rep(0,trun-q))
    # set psi_j = 0 for j < 0
    psi.0 <- numeric(trun+p)
    psi.0[p] <- 1 # this is psi_0
    for(j in 1:trun)
    {
      psi.0[p+j] <- theta.0[j] + sum( phi[1:p] * psi.0[(p+j-1):j] )
    }
    # take away zeroes at beginning
    psi <- psi.0[p:(p+trun)]
  }
  return(psi)
}
```

E Python code

```
# NB! Heavily inspired by: https://www.analyticsvidhya.com/blog/2021/05/
#       bitcoin-price-prediction-using-recurrent-neural-networks-and-lstm/
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
```

Preprocessing: We start by reading the downloaded time series with Pandas. The data has invalid price values, indicated by ".", so we have to remove these and impose the prices to be numeric again.

```
df = pd.read_csv('../Tidsrekker/CBLTCUSD.csv', date_parser = True)
# Sort out invalid price
df['CBLTCUSD'] = df['CBLTCUSD'][~(df['CBLTCUSD'] == '.' )]
df['CBLTCUSD'] = pd.to_numeric(df['CBLTCUSD'])

df.tail()
```

Here, we split the data in a training and testing DataFrame. We also drop the actual dates from the training data.

```
n = len(df)
train_df = df[0:int(n*0.9)]
test_df = df[int(n*0.9):]

df_train = train_df.drop('DATE', axis=1)
test_df.head()
```

The data is normalized with Keras' MinMaxScaler so it's ready to be passed into the network.

```
scaler = MinMaxScaler()
df_train = scaler.fit_transform(df_train)

# Pass inputs parameters properly to be inserted to model
X_train = []
Y_train = []
for i in range(60, df_train.shape[0]):
     X_train.append(df_train[i-60:i])
     Y_train.append(df_train[i,0])
X_train, Y_train = np.array(X_train), np.array(Y_train)

X_train.shape
```

Building the model: Here we state the LSTM model. It has 4 layers with one droput and one Dense layer

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
# Initialize RNN
model = Sequential()
model.add(LSTM(units = 50, activation = 'relu', return_sequences = True,
↪   input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units = 60, activation = 'relu', return_sequences = True))
# model.add(Dropout(0.3))
model.add(LSTM(units = 80, activation = 'relu', return_sequences = True))
# model.add(Dropout(0.4))
model.add(LSTM(units = 120, activation = 'relu'))
# model.add(Dropout(0.5))
model.add(Dense(units =1))
model.summary()
```

Fitting the model

```
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
NUM_EPOCH = 20
history= model.fit(X_train, Y_train, epochs = NUM_EPOCH, batch_size = 128,
↪   validation_split=0.20)
```

We find the loss for each epoch

```
loss = history.history['loss']
epochs = range(len(loss))
```

```
plt.figure()
plt.plot(epochs, loss, 'b', label='Training loss')
# plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title("Training loss, " + str(NUM_EPOCH)+ " epcohs")
plt.legend()
plt.savefig("Train_loss_" + str(NUM_EPOCH) + "_epochs.pdf")
plt.show()
```

Testing the fit of the model:

```
part_60_days = train_df.tail(60)
# df2= part_60_days.append(test_df, ignore_index = True) # This one to exclude
 ↪  the newest values

df_test_new = pd.read_csv("../Tidsrekker/CBLTCUSD_testonly.csv", date_parser =
 ↪  True)
df_test_new['CBLTCUSD'] = pd.to_numeric(df_test_new['CBLTCUSD'])

df2 = part_60_days.append(df_test_new, ignore_index = True)
df2 = df2.drop(['DATE'], axis = 1)
df2.head()

inputs = scaler.transform(df2)

X_test = []
Y_test = []
for i in range (60, inputs.shape[0]):
    X_test.append(inputs[i-60:i])
    Y_test.append(inputs[i, 0])
X_test, Y_test = np.array(X_test), np.array(Y_test)
X_test.shape, Y_test.shape

Y_pred = model.predict(X_test)

scale = 1/scaler.scale_
Y_test = Y_test*scale
Y_pred = Y_pred*scale
```

Determine fit for the last 21 days only:

```
# This one is just to see how it looks like with the newest 21 days only
plt.figure(figsize=(14,5))
plt.plot(Y_test[-21:], color = 'red', label = 'Real Litecoin Price')
plt.plot(Y_pred[-21:], color = 'green', label = 'Predicted Litecoin Price')
plt.title('Price Prediction of newest values (' + str(NUM_EPOCH) + ' epochs)')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.savefig("NEWONLY_LSTM-pred-" + str(NUM_EPOCH) + ".pdf")
plt.show()

latest_21_mean_error = np.mean(np.abs(Y_test[-21:] - Y_pred[-21:]))
test_mean_error = np.mean(np.abs(Y_test[-204:] - Y_pred[-204:]))
print("Latest 21 entries:", latest_21_mean_error)
print("All test entries:", test_mean_error)
```

For all test data:

```
plt.figure(figsize=(14,5))
plt.plot(Y_test, color = 'red', label = 'Real Litecoin Price')
plt.plot(Y_pred, color = 'green', label = 'Predicted Litecoin Price')
plt.title('Litecoin Price Prediction using RNN-LSTM with ' + str(NUM_EPOCH) + '
↪  epochs')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.savefig("LSTM-pred-" + str(NUM_EPOCH) + "-epochs.pdf")
plt.show()
```

Bibliography

[1] Contentworks Agency. *A brief history of Litcoin*. Accessed: 18th Oct. 2021. 2019. URL: https://contentworks.medium.com/a-brief-history-of-litecoin-260c96faa454.

[2] Peter Brockwell and Richard Davis. *An Introduction to Time Series and Forecasting*. Vol. 39. Jan. 2002. ISBN: 978-1-4757-2528-5. DOI: 10.1007/978-1-4757-2526-1.

[3] Jason Brownlee. *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. 2019. URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (visited on 20th Oct. 2021).

[4] CoinHouse. *Litecoin Price*. Accessed: 18th Oct. 2021. 2020. URL: https://www.coinhouse.com/litecoin-price/.

[5] A. Graves and J. Schmidhuber. 'Framewise phoneme classification with bidirectional LSTM networks'. In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 4. 2005, 2047–2052 vol. 4. DOI: 10.1109/IJCNN.2005.1556215.

[6] María Isabel and Fraga Alves. 'Statistical Inference for Heavy and Super-Heavy-tailed distributions'. In: 2005.

[7] Jana Kane. *Litecoin Price Predictions: How it could go in 2021 and beyond*. Accessed: 18/10-2021. 15th Oct. 2021. URL: https://www.liteforex.com/blog/analysts-opinions/litecoin-price-prediction-forecast/.

[8] Uday Sampath Kumar. *Walmart says looking into fake press release on litecoin tie-up*. Accessed: 18/10-2021. 14th Sept. 2021. URL: https://www.reuters.com/business/retail-consumer/press-release-walmarts-litecoin-partnership-is-fake-cnbc-2021-09-13/.

[9] TME News. *Will Litecoin set a new all-time high?* Accessed: 18/10-2021. 2021. URL: https://themunicheye.com/will-litecoin-set-a-new-all-time-high%3F--4204.

[10] Richard A. Davis Peter J. Brockwell. *Time Series: Theory and Methods*. Apr. 2009, p. 245. ISBN: 9781441903198. DOI: 10.1007/978-1-4419-7865-3.

[11] Machael Phi. *Illustrated Guide to Recurrent Neural Networks*. 2018. URL: https://s.ntnu.no/bbrbqRJM (visited on 19th Oct. 2021).

[12] Litecoin Project. *Litecoin - The Cryptocurrency for payments*. Accessed: 18th Oct. 2021. 2011. URL: https://litecoin.org/.

[13] Robert Shumway and David Stoffer. *Time Series Analysis and Its Applications With R Examples*. Vol. 9. Jan. 2011. ISBN: 978-1-4419-7864-6. DOI: 10.1007/978-1-4419-7865-3.